**Networking Assignment – Group 13**

Group Members

* Rachael Espie
* Magdalena Veleva
* Silvi Koycheva
* Aleksandra Kliaugaite

**Part 1**

1. Formulas Used:

Raw Data Rate = (Video Resolution)x(Bits per Pixel)x(Frame Rate)x(3)x(Seconds)

Compressed Video Data = Raw Data Rate / Compression Rate

Total Overhead per Packet = 7 x Header Data (bits)

Total Packets Needed for Payload = (Compressed Video Data x 10⁹ (converting to bits))/Packet Length

Total Bits Needed to Carry Overhead Information = Total Overhead per Packet x Total Packets Needed for Payload

Total Number of Bits = Compressed Video Data + Total Bits Needed to Carry Overhead Information

Physical Layer Bit Rate Required for Live Streaming = Total Number of Bits/(Video Length in Seconds)

Calculations:

Raw Data Rate = (3,840 x 2,160) x 12 x 100 x 3 x (100 x 60) = 179,159,040,000,000 = 179,159.04 Gbits

Compressed Video Data = 179,159.04 / 250 = 716.64 Gbits

Header Data in Bits = 30 \* 8 = 240 bits

Total Overhead per Packet = 7 \* 240 = 1,680 bits

Total Packets Needed for Payload = (716.64\*109)/10,000 = 71,664,000 bits

Total Bits Needed to Carry Overhead Information = 1,680 \* 71,664,000 = 120,395,520,000 bits = 120.4 Gbits

Total Number of Bits = 716.64 + 120.4 = 837.04 Gbits

Physical Layer Bit Rate Required for Live Streaming = 837.04/(100\*60) = 0.1395 Gbits/s

= 139.5 Mbits/s

Data Rate After Channel Coding = 3/2\*139.5 **= 209.25 Mbits/s**

1. Formulas Used:

QAM Symbol Rate = 1/4 x Channel Coding x Total Number of Bits for Live Streaming

Calculations:

16 QAM = 2⁴ QAM

1/4\*3/2\*139.5 **= 52.31 Msymbols/s**

1. No, we cannot stream as the required bandwidth - 52.31 is above the available bandwidth - 40 Msymbols/s
2. Formulas Used:

File Size in Msymbols = Symbol Rate \* Transmission Time in Seconds

Download Time = File Size / Channel Bandwidth

Total Download Time = Download Time + Latency

Calculations:

Let’s assume we have a UDP connection. If we can stream in real time then the download time would be 100 min - the length of the video. But we can’t.

File Size in Msymbols = 52.3143 \* 100 \* 60 = 313,885.8 Msymbols

But we don’t have 52.3143 Msymbols / s available. We have channel bandwidth of 40  Msymbols / s available and we can’t stream the video in real-time.

That’s why for TCP:

Download time = = 313,885.8 / 40 = 7,847.145 s = 130.79 mins

We should also add the latency to get the total download time.

Total Download Time = 7,847.145 s + 0.13 s = 7,847.275 s **= 130.79 mins**

1. Formulas Used:

TCP Throughput = (TCP Window Size Bits/ Round Trip Latency)/Sqrt(PLR)

Round Trip Latency = (Total Distance between London/Tokyo\*2 IN METRES (\*1000)) /Speed of Cable

Bandwidth in Bit/s = Bandwidth in Msymbol/s \* Bits per Symbol

TCP Window Size in Bits = Bandwidth in Bit/s \* Round Trip Latency in Seconds

Calculations:

PLR is the % packet loss rate = 5%

Round Trip Latency = 26000x10³ / 2x10⁸ = 0.13 seconds = 130 ms (milliseconds)

Bits per Symbol = 4 because 2⁴ = 16 which is the QAM we are using

Bandwidth in Bits = = 40\*4 = 160 Mbits/s = 160 000 000 bits/s

TCP Window Size in Bits = 160 \* 10⁶ \*0.13  = 20.8 \* 10⁶ bits  = 20.8 Mbits

TCP Throughput = (20.8/0.13)/Sqrt(5) = **71.55 Mbits/s**

1. The packet size is 10,000 bits. This is quite high especially for using in an error-prone environment such as a mobile application. If a single packet is lost then we also lose 10,000 bits which will significantly impact on the streaming of the video. This could lead to frames getting lost, which can cause the video to stutter instead of flowing smoothly which is desired. If the lost frames are I Frames then this may not be as noticeable as there are no dependencies between I Frames, so once the next frame appears this error will disappear. However if this error occurs in a P Frame then this error may propagate between frames and will spread around the frames due to motion vectors. This will cause a very bad visual quality which will deteriorate greatly until a new refresh frame arrives. In error-prone environments it is better to use smaller packet sizes so that if a packet is lost then minimal data is lost as well.

In communications point of view larger packet size will have a major impact on receiver quality. Difficult to conceal the lost information when the packet size is larger under high PLR.

1. As the proposed network sends the majority of its data through fibre optic, it would dramatically improve the throughput if the QAM was increased to 128 QAM or 256 QAM, this would increase the bits per symbol from 4 to 7-8 bits per symbol.

Channel coding is used for detecting errors during transmission, which is needed in high interference environments as errors are more likely. If we increase the channel coding rate we will reduce the amount of redundant data we are sending, and increase the likely hood of detecting errors. We should increase the channel coding from 2/3 to 3/4 to make it more efficient.

Increasing the video compression rate will mean that the raw data size is reduced. Ideally we want to optimise lossless compression as much as possible as this will reduce the data without affecting the video quality. Lossy compression can also be used if the video quality is less important.

If we change to a wireless communication channel then we should make the following modifications:

* 16 QAM when conditions are good, 8 QAM when conditions are poor
* Increase channel coding rate
* Increase video compression rate

As the QAM Modulation Order increases the bit rate/throughput increases but it also reduces resistance to errors. Many over-the-air systems (satellite, cellular) dynamically adapt modulation order based on channel conditions, so that when conditions are good and interference is low then we use a higher order, and fall back to lower order when conditions are poor. 16 QAM is commonly used for satellite channel communication but in order to prevent errors 8 QAM could be used when conditions are poor.

As a wireless environment experiences a large amount of noise and interference we should use a higher channel coding rate. This will make it easier to spot and recover from errors.

Increasing the video compression rate will reduce the raw data size and as such increase the chances of real time streaming being supported.

**Part 2**

**Stage 1**

1. Your packet format and an explanation of all the fields.

We have added a Sender and a Receiver class which extend the TransportLayer class. The Sender is responsible for passing packets from the Application layer to the Network layer. The Receiver is responsible  for passing packets from the Network layer to the Application layer.

Our packet is formatted using the following fields: seqnum and acknum which are integers, checksum which is of type byte and data which is an array of bytes. We did add setter and getter methods for all fields as well as added a method for calculating checksum.

Seqnum – this stores the sequence number of the packet which is either 0 or 1 and is used by receiver to check if the received packet is the one we are currently waiting for.

Acknum – this stores the acknowledgment number of the packet which is either 0 or 1. The receiver sends this acknum back to sender to indicate whether the packet was received successfully or not.

Checksum – we have a field that stores the checksum and a method that calculates the checksum value for the given data, and is used by both the sender and the receiver to check if the data has been corrupted in any way. It is calculated by summing all the bytes in data and then multiplying this by 0xFFFFFFFF

Data – this stores the data that is being contained and sent with the packet. We receive this data from the Network Simulator and add this into a packet when rdt\_send in the sender is called.

1. Explain all situations that can occur when sending packets and how you handle them.

* **No Loss, delay or corruption** 
  + When the packet has not been lost or corrupted the Sender starts the timer and sends the packet to the Receiver and the Receiver sends it to the application layer and sends an acknowledgement to the Sender. When the Sender receives the right acknowledgement it stops the timer and sends a new packet.
* **Packet corruption** 
  + We check if the packet is corrupted in both the sender and receiver by comparing the checksum of the data. If we have a corrupted packet in the sender we don’t send it and instead wait for a timeout to occur. When we have a corrupted packet in the receiver we don’t send it to the application layer but instead we send back the packet with an acknowledgement number different from the one the Sender is waiting for. In this case a timeout occurs and the timerInterrupt() method resends the last sent packet.
* **Packet delay** 
  + When there is a packet delay exceeding the increment time set when starting the timer then there is timeout which triggers the timerInterrupt() method and the last sent packet will be resent.
* **Packet loss when sending data** 
  + In this case the Receiver doesn’t receive a packet so it doesn’t send an acknowledgement to the Sender. A timeout occurs in the Sender and the timerInterrupt() method resends the last sent packet.
* **Packet loss when sending acknowledgment** 
  + In this case the Sender doesn’t receive a packet with acknowledgement so a timeout occurs and the timerInterrupt() method resends the last sent packet.

1. Post screenshots of each situation being handled and include the code fragments that handle them.

**Packet Corruption**

**Text

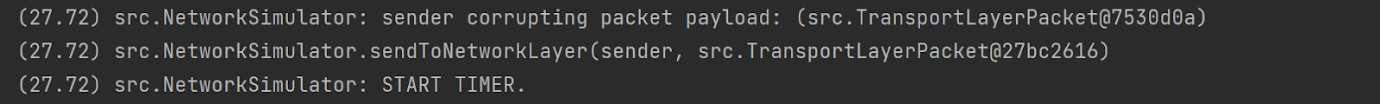
Description automatically generated**

**Text

Description automatically generated**

**Text

Description automatically generated**

****

**Acknum Corruption**

**Text

Description automatically generated**

**Text

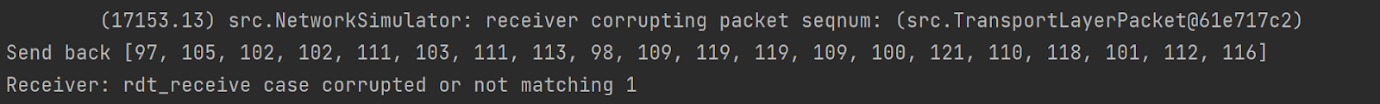
Description automatically generated**

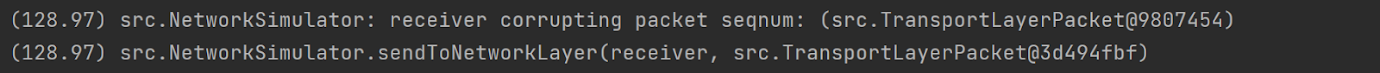
Text

Description automatically generated

**Seqnum Corruption**

**Text

Description automatically generated**



**Packet Delay**

**Graphical user interface, text

Description automatically generated**

**Text

Description automatically generated**

**Text

Description automatically generated**

**Packet Loss when Sending Data**

**Graphical user interface, text

Description automatically generated**

**Text

Description automatically generated**

**Text

Description automatically generated**

**Packet Loss when Sending Acknowledgement**

**Graphical user interface, text

Description automatically generated**

**Text

Description automatically generated**

1. How you tested your code? (Note: I’m not looking for extensive JUnit tests but tests. to show your protocol works for all situations)

We tested our code by including clear print statements in nearly all methods, allowing us to see what data, checksum, and sequence/acknowledgement number was transmitted and received. In the Network Simulator instance, we kept altering the corrupt loss probabilities to check how our application dealt with corrupted packets and if it processed them correctly.

**Stage 2**

1. Explain how you handled pipelining.

We implemented Go Back N pipelining, which means that the sender can send many packets without waiting for an acknowledgment, but there can only be N unacknowledged packets in the window.

**Sender**

**rdt\_sent()**

The fields base and nextSeqnum have been added to our sender; base stores the oldest unacknowledged packet number, and nextSeqnum indicates the sequence number of the next packet to be delivered. When we construct a packet in rdt\_send() in sender, we pass nextSeqnum for acknum and seqnum parameters. If there are no more available slots in the window for the next packet, we store it in a queue to be dispatched when one becomes available.

**rdt\_receive()**

If the received packet is not corrupted then we set base to be this packet’s acknum+1 this is because all packets before this can be acknowledged due to cumulative acknowledgements. If base is equal to nextSeqnum then this means we have no packets in window so we stop the timer. Otherwise we stop the current timer and start a new one. If we have packets stored in our queue we call rdt\_sent() on them so they are sent in the right order.

**Receiver**

**rdt\_send()**

If the packet is not null and its sequence number is equal to the expected one we send this data to the application layer.

**rdt\_receive()**

If the packet is not corrupted and its seqnum matches our expected seqnum we can then send it to the application layer. Otherwise we create a new packet with the same seqnum and data and an acknum set to expectedSeqnum -1 and send this to the network layer, so the sender can resend the correct non-corrupted packet. expectedSeqnum -1  is the number of the last acknowledged packet.

1. Explain steps 1-4 from above for pipelining. If there is no change to a specific part just state that it is the same
2. Your packet format and an explanation of all the fields.

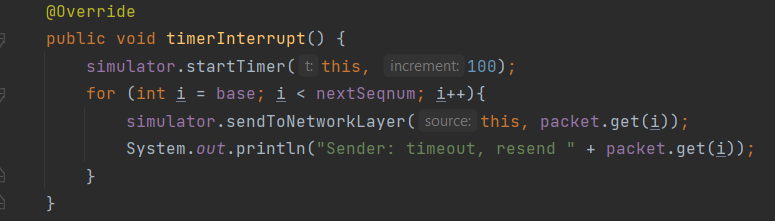
It is the same as for stage 1. The only difference is that we have only one constructor.

1. Explain all situations that can occur when sending packets and how you handle them.

* Timeout event - If the sender encounters a timeout, all packets that have been sent but not yet acknowledged are resent. The timer is restarted when an ACK is received but there are still further transmitted but unacknowledged packets. The timer is ended if there are no pending, unacknowledged packets.
* Receiving out-of-order packets - When the receiver receives out-of-order packets, instead of buffering them, it simply discards them. There is no point in buffering them because they will be retransmitted at the sender due to the GBN retransmission policy.
* Invocation from above when the window size is full - in this case we save the incoming packets in a queue in Sender and send them once the window size is not full
* Invocation from above when the window size is not full - a packet is created and sent to the Receiver. In Receiver if the packet is not corrupted and the sequence number is the same as the expected one the packet is sent to the application layer and expectedSeqnum is increased by 1. If the sender receives a non-negative acknowledgement it updates its base

1. Post screenshots of each situation being handled and include the code fragments that handle them

**Testing lost packets**



A picture containing calendar

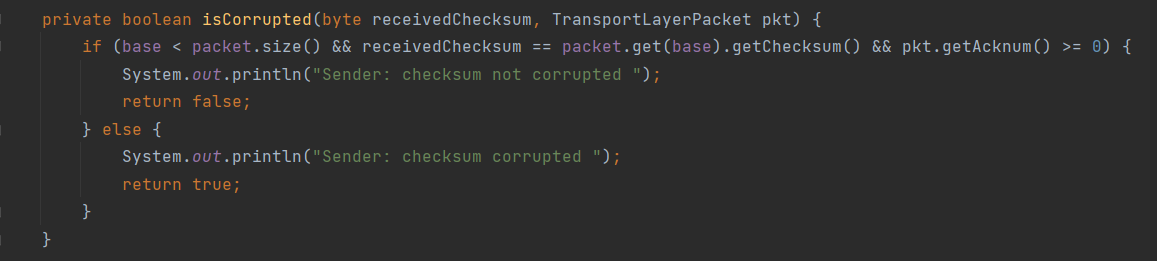
Description automatically generated

Text

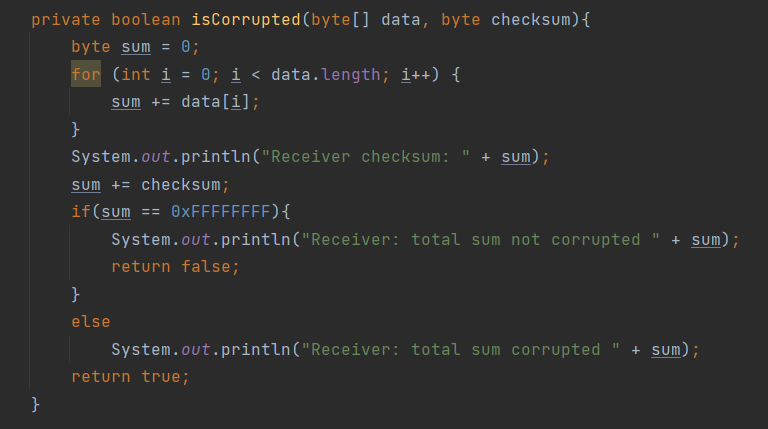
Description automatically generated

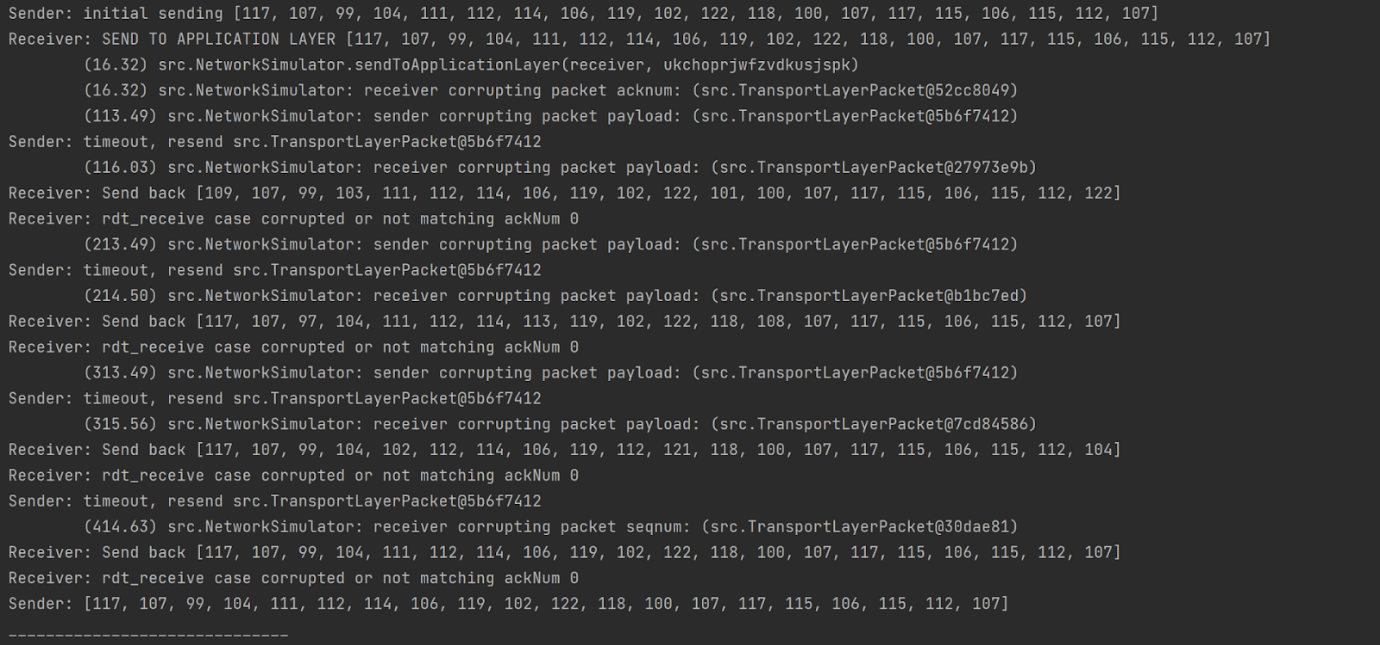
**Testing Corrupted Packets, Seqnum & Acknum**

**Sender:**

****

**Receiver:**

****



Text

Description automatically generated

**Invocation from above when the window size is full**

****

****

**A picture containing calendar

Description automatically generated**

**Invocation from above when the window size is not full**

****

A computer screen capture

Description automatically generated with medium confidence

1. How you tested your code?

We tested the code the same way as we did for stage 1.

**Workload Report**

1. State of functionality your program achieved.

We have done everything starting from rdt 1.0 and progressing to rdt 3.0 which is our final version using a stop-and-wait protocol. Then we completed the pipeline using Go-Back-N.

1. Details of the tasks involved in completing the project

* Creating and implementing two additional classes called Sender and Receiver that extend Transport Layer class following the diagrams provided by the book.
* Implementing Transport Layer Packet class.
* Testing code
* Writing the report

1. What team members completed what tasks?

Part 1 calculations were completed by all team members, and our results were compared during the zoom session. Part 2 involved us working together on the stages and discussing problems that we should try to solve in our spare time. Everyone in the group worked on deliverable 2, and we shared a Google Docs file so that we could edit the report in real time. All of us attempted implementing stage 1 and stage 2 from part 2 and during our meetings we have been putting together the solution. Everyone contributed by fixing bugs, suggesting improvements and testing during our Zoom meetings.

1. Summary of how we worked together as a team

We had 1-2 weekly zoom sessions where we tested code, resolved bugs, clarified topics that were obscure to each other, and discussed what tasks we had accomplished so far and what still remained to be completed. In addition, while working on this project, we had a group chat on Messenger where we communicated on a daily basis

1. Lessons learned and what you would do differently next time

We have learned how to manage our time, operate as a team, communicate effectively, form friendships with our teammates, and assist one another when we encounter difficulties. Completing this exercise improved our comprehension of computer networking topics while also introducing fresh approaches to java code implementation. Next time we would try to have more sessions together as we spot errors more easily and we are progressing faster on the project.